COMSM1302 電腦架構概論

Week 7 作業：The Hack CPU 完整詳解

在 Logisim 上打造能執行 Hack 機器碼的完整電腦

完整建造指南（指令解碼 · 資料通路 · 控制訊號 · 測試與除錯）

2026 年 7 月

本作業的任務：

1. 在 Logisim 上打造一台能執行 Hack 的電腦！
2. 剩餘時間拿去多練組語——課程接下來的一切都建立在組語上，而本週作業主要只與本週教材相關（開始前至少應完成第五週作業第 5 節）。

素材盤點：skeleton 檔已含第二週的 ALU 與第四週的程式計數器；RAM、ROM、暫存器用 Logisim 預建版。影片 2 的架構輪廓**不是完整解答**——填補空隙正是本作業的核心工作，也是本詳解逐步示範的內容。

目錄

1　規格整理：我們要蓋什麼?

題目

把「Hack 電腦」的規格整理成可以動工的藍圖：需要哪些元件？指令如何編碼？記憶體為什麼用 64KB ／ 15 位元位址而不是「真正的」32KB ／ 14 位元?

解答

整機藍圖（Harvard 架構）：指令與資料**分開存放**——這是 Hack 能用單一時脈週期執行每條指令的關鍵。

元件	來源	規格
ALU	第二週作業（skeleton 內附）	16 位元、zx, nx, zy, ny, f, no 控制、zr, ng 旗標
PC	第四週作業（skeleton 內附）	16 位元、reset > load > inc 優先序
A、D 暫存器	Logisim memory 函式庫 Register	16 位元、時脈上升緣載入
ROM	Logisim memory 函式庫	位址 15 位元、資料 16 位元、存程式
RAM	Logisim memory 函式庫	**位址 15 位元**（64KB）、資料 16 位元

為什麼 RAM 用 15 位元位址 (64KB)? 真正的 Hack 只有 32KB RAM（14 位元位址，`0x0000`–`0x3FFF`），螢幕與鍵盤是 **獨立的記憶體映射裝置**（`0x4000`–`0x6000`）。本課程是架構課不是硬體設計課，不直接模擬螢幕鍵盤——把 RAM 直接加大到 15 位元位址空間，讓 `0x4000`–`0x6000` 這段位址**落在普通 RAM 裡**：程式照常讀寫「螢幕」與「鍵盤」，只是沒有真的畫面；想模擬按鍵就暫停模擬、手動改 `RAM[0x6000]` 再繼續。

指令編碼（本週影片的核心，動工前先背熟）：

型態	位元 15	格式
A-指令	`0`	`0vvvvvvvvvvvvvvv`　（v = 15 位元常數）
C-指令	`1`	`111` $a\, c_1c_2c_3c_4c_5c_6\, d_1d_2d_3\, j_1j_2j_3$

C-指令的欄位對應（**位元編號**是除錯時的救命索）：

欄位	指令位元	意義
型態	i_{15}	0 = A-指令、1 = C-指令
a	i_{12}	ALU 的 y 輸入：0 = A、1 = M
$c_1..c_6$	$i_{11}..i_6$	直通 ALU 的 zx, nx, zy, ny, f, no
$d_1\, d_2\, d_3$	$i_5\, i_4\, i_3$	寫入 A ／ D ／ M
$j_1\, j_2\, j_3$	$i_2\, i_1\, i_0$	結果 <0 ／ $=0$ ／ >0 時跳躍

未定義行為的經驗法則（題目原話）：規格沒說的情況，你的 CPU 做什麼都可以。例如 A = `0xFFFF` 時執行 `D=M` 的結果無所謂——M 只在 A 是合法位址時有定義。工程意義：**不要為不存在的情況多蓋硬體**（RAM 位址直接取 A 的低 15 位、忽略最高位即可）。

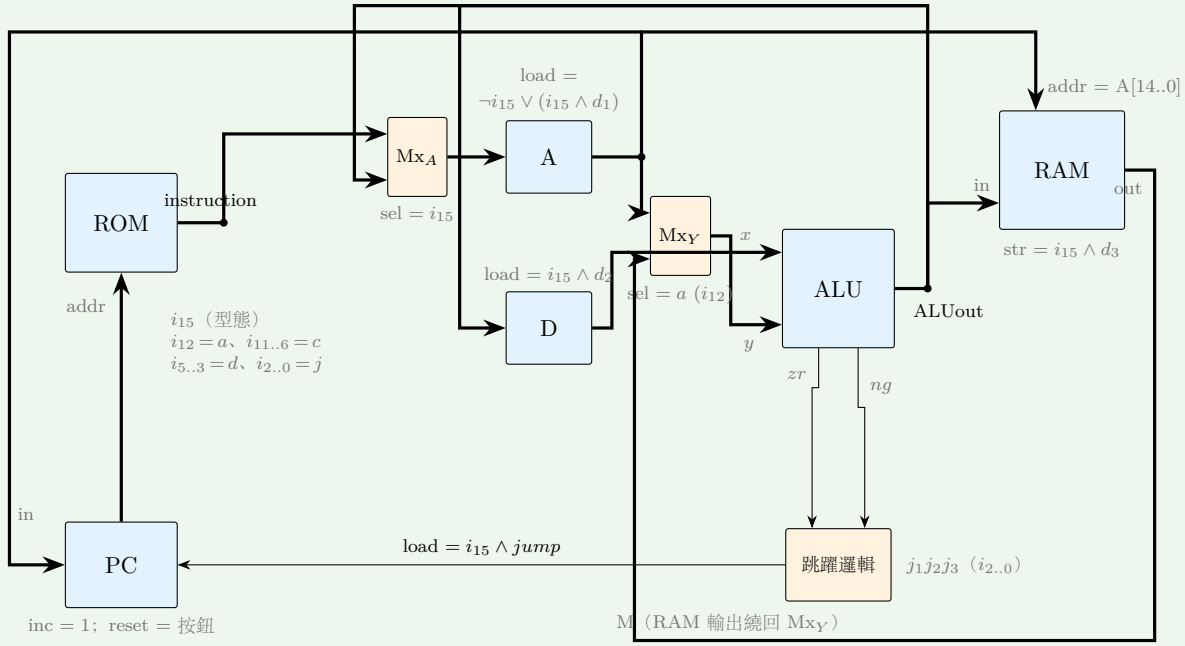
2　資料通路設計（填補影片 2 的空隙）

題目

畫出完整的資料通路，並為每個元件推導控制訊號：A 暫存器什麼時候載入、載入什麼？D 呢？RAM 的寫入使能？ALU 的輸入從哪來？PC 什麼時候跳躍？

解答

整機資料通路：



（粗線 = 16 位元匯流排；細線 = 1 位元控制訊號。）

控制訊號逐一推導（這就是「影片 2 沒給的空隙」）：

1. **Mx$_A$（A 的來源）**：A-指令要把 **指令本身**裝進 A（最高位是 0，整條 16 位元直接接進去就等於零延伸的 15 位元常數）；C-指令若寫 A（`A=...`）裝的是 ALU 輸出。

$$\text{sel} = i_{15}, \qquad \text{in}_0 = \text{instruction}, \quad \text{in}_1 = \text{ALUout}$$

2. **A 的載入**：A-指令**必定**載入；C-指令只在 d_1（i_5）時載入：

$$\boxed{A.\text{load} = \neg i_{15} \vee (i_{15} \wedge i_5)}$$

3. **D 的載入**：只有 C-指令的 d_2（i_4）：

$$\boxed{D.\text{load} = i_{15} \wedge i_4}$$

   必須用 i_{15} 把關——不然 A-指令 `@21`（位元圖案裡恰好 $i_4 = 1$ 的常數，例如 `@16`）會誤寫 D！這是本作業第一大坑。

4. **RAM**：位址永遠是 A 的低 15 位；寫入資料永遠是 ALUout；寫入使能只有 C-指令的 d_3（i_3）：

$$\boxed{\text{RAM.str} = i_{15} \wedge i_3}, \qquad \text{RAM.addr} = A[14..0]$$

5. **ALU**：$x = D$ 固定；y 由 a 位（i_{12}）在 A 與 M（RAM 輸出）之間選；六條控制線**按順序直通**：

$$(zx, nx, zy, ny, f, no) = (i_{11}, i_{10}, i_9, i_8, i_7, i_6)$$

（順序接反是本作業第二大坑——c_1 是 **高位** i_{11}。）

6. **跳躍邏輯**：ALU 附贈兩面旗子 zr（結果 $= 0$）與 ng（結果 < 0），「結果 > 0」$= \neg zr \wedge \neg ng$：

$$jump = (j_1 \wedge ng) \vee (j_2 \wedge zr) \vee \big(j_3 \wedge \neg zr \wedge \neg ng\big)$$

$$\boxed{PC.\text{load} = i_{15} \wedge jump}, \qquad PC.\text{in} = A$$

再度用 i_{15} 把關：A-指令絕不跳躍（不然 `@7` 這種常數會被當成 `JMP`——第三大坑）。

7. **PC 的 inc**：第四週的 PC 優先序是 reset > load > inc，所以 **inc 可以恆為 1**——不跳躍、不重設時自然 +1；reset 接一顆按鈕（開機／重跑）。

一個時脈週期的生命：上升緣 → PC 輸出位址 → ROM 吐出指令 → 解碼（splitter 拆位元）→ ALU 組合運算（讀 A、D、M 都是舊值）→ 下一個上升緣把結果**同時**寫進 A ／ D ／ M ／ PC。**取指與執行在同一個週期完成**——Harvard 架構讓指令記憶體與資料記憶體互不打架，這就是 Hack 能單週期的原因。

3　Logisim 實作細節

題目

把第 2 節的藍圖搬進 Logisim：元件屬性怎麼設？指令位元怎麼拆？時脈怎麼接？

解答

(1) 指令解碼：一顆 splitter 搞定。 ROM 輸出接 16 路 splitter（Fan Out = 16，或按欄位分組：`15`、`12`、`11-6`、`5-3`、`2-0`）。建議在每條拆出的線上貼 Logisim 的 Text 標籤（a、c_1、$\cdots$）——兩週後回來除錯時你會感謝自己。控制邏輯只需要幾顆小閘：

- 1 顆 NOT（$\neg i_{15}$）、1 顆 OR、3 顆 AND 做 A/D/RAM 的載入訊號；
- 跳躍邏輯：2 顆 NOT、4 顆 AND、1 顆三輸入 OR （或兩顆二輸入）。

(2) RAM 屬性（wiring 好之前先設好）：

- Address Bit Width = 15、Data Bit Width = 16；
- **Data Interface = “Separate load and store ports”**——讀寫分開、不用三態緩衝器，東側輸出（load 腳恆 1）隨時反映 `RAM[addr]`，恰好是 Hack 要的「M 隨時可讀」；西側輸入接 ALUout，`str` 接 $i_{15} \wedge d_3$、時脈接全域時脈；
- ROM 同理：Address 15、Data 16，位址接 PC 低 15 位。

(3) 時脈與重設： A、D、PC、RAM 全部吃**同一個**時脈。Logisim 小技巧（題目原話）：電路裡可以放**多個**時脈輸入，“Simulate → Tick once”（Ctrl+T）會讓它們**一起**走一步——比拉一條蜘蛛網般的時脈線到每個角落乾淨得多。reset 按鈕接 PC 的 reset；想重跑程式：按住 reset、tick 一次、放開。

(4) 兩個容易忽略的接線細節：

- A-指令進 Mx_A 時**整條 16 位元直接接**——不需要特別做「零延伸」，因為 A-指令的 i_{15} 本來就是 0；
- RAM 位址接 A[14..0]（splitter 丟掉最高位）——A = `0xFFFF` 時會讀寫 `RAM[0x7FFF]`，沒關係：ISA 沒定義這個情況，經驗法則說隨便它。

4　測試你的 CPU

題目

先測幾條單獨的指令，再載入夠複雜的程式做壓力測試（skeleton 的 ROM 已預載上週 Collatz 題的解答），必要時與 CPU 模擬器並排對照。

解答

階段一：單指令煙霧測試。手工組譯幾條指令、直接填進 ROM（右鍵 → Edit contents），每 tick 一次核對暫存器。建議的最小測試集（十六進位已算好，直接貼）：

指令	機器碼（hex）	預期效果（tick 後）
`@21`	`0015`	A = 21（測 A-指令載入）
`D=A`	`EC10`	D = 21（測 ALU 傳 A、寫 D）
`@16`	`0010`	A = 16 （**D 必須不變**——驗證 i_{15} 把關！）
`M=D`	`E308`	RAM[16] = 21（測寫 M）
`D=D+M`	`F090`	D = 42（測 a 位選 M）
`M=M+1`	`FDC8`	RAM[16] = 22（測讀改寫 M）
`D;JGT`	`E301`	PC 跳到 A = 16 （D> 0；測跳躍）
`0;JMP`	`EA87`	PC = A（測無條件跳）

檢查點（每條指令後看三個地方）：PC 是否 +1（或跳對地方）、目標暫存器是否更新、**其他**暫存器是否**沒有**被誤寫。第三點最容易抓到控制訊號忘了用 i_{15} 把關的 bug。

階段二：Collatz 壓力測試。 skeleton 的 ROM 已預載上週 Collatz 解答：

1. 在 RAM 編輯器把 `RAM[0]` 設成 `0x0005`，開自動 tick（Simulate → Ticks Enabled，頻率調最高）；
2. 預期 3–4 秒跑完；`RAM[0x20]` 起應出現軌跡 5, 16, 8, 4, 2, 1（hex：`5, 10, 8, 4, 2, 1`）；
3. 再試 `RAM[0]` = `0x002C`（44），預期約 10 秒，軌跡 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1；

4. **跑太久 = 出事了**（題目給的基準）。最常見原因：跳躍邏輯錯（無窮迴圈跳不出去）或 PC 的 load/inc 優先序接反。

並排除錯法：同一支程式開在 nand2tetris CPU 模擬器裡，兩邊同步單步——第一個分歧的週期就是 bug 所在；分歧時看四個值：PC、A、D、剛寫入的 RAM 格。

階段三：載入自己的程式（bin2hex 流程）。

1. 用 nand2tetris 組譯器把 `.asm` 編成 `.hack`（0/1 文字檔）；
2. 命令列跑 `bin2hex` 輸入`.hack` 輸出`.txt` ——把每個二進位字轉成以空白分隔的十六進位字（非 Windows 請自行編譯它的 C 原始碼：`cc bin2hex.c -o bin2hex`）；
3. 文字編輯器開輸出檔、全選複製；
4. Logisim 中右鍵 ROM → “Edit contents...”；
5. 游標點到位址 0、Ctrl+V（Mac：Cmd+V）貼上；
6. 關閉視窗，程式已進 ROM。

模擬鍵盤輸入（題目提供的方法）：暫停模擬 → 直接編輯 `RAM[0x6000]` 為想要的掃描碼（`c` 鍵 = 99 = `0x63`）→ 繼續模擬。放開按鍵就改回 0。螢幕同理：程式寫進 `0x4000`–`0x5FFF` 的值可以在 RAM 編輯器裡直接觀察（沒有畫面，但位元圖案就在那裡）。

5 常見錯誤型錄（除錯速查）

解答

按「症狀 → 病因」整理本作業最常見的八種 bug：

症狀	病因
`@16` 之類的 A-指令把 D 或 RAM 改掉	D ／ RAM 的載入訊號忘了 $\wedge i_{15}$（常數的 i_4 ／ i_3 位恰好是 1）
A-指令後 PC 亂跳	跳躍訊號忘了 $\wedge i_{15}$（常數的低 3 位恰好非零，被當 jump 欄位）
`D=A` 對、`D=M` 錯（或相反）	Mx$_Y$ 的 select 接反（$a=1$ 應選 M）
`D+1`、`D-A` 等個別 comp 錯	$c_1..c_6$ 與 $zx..no$ 順序接反（c_1 是**高位** i_{11}！）
`JGT` 在結果為負時也跳	「>0」少算 $\neg zr \wedge \neg ng$（只接了 $\neg ng$，0 也會跳）
每條指令 PC 都不動（或跳躍後又 +1）	PC 的 load/inc 優先序錯、或 inc 沒接 1
`M=M+1` 少加或跳拍	RAM 讀取不是組合式（Data Interface 沒選 separate ports / 非同步讀），M 舊值晚一拍
Collatz 跑超過 30 秒	以上任一；並排單步找第一個分歧週期

進階討論

你剛蓋好的是「單週期微架構」。每條指令恰好一個時脈週期，週期長度必須容納**最長**的組合路徑：ROM 取指 → 解碼 →（可能經 RAM 讀 M）→ ALU → 跳躍邏輯 → PC——這條 **關鍵路徑** 決定時脈上限。真實 CPU 的做法是**管線化**（第七週講義後半）：把這條路切成取指／解碼／執行／訪存／寫回五段，五條指令同時在不同段流水——吞吐量升五倍，但也帶來危障（hazard）要處理。Hack 能單週期的兩大功臣：（1）**Harvard 架構**——取指與資料訪問用不同記憶體，不搶埠；（2）指令**定長且解碼極簡**——欄位直通 ALU，不需要微碼。對照 x86：變長指令、上千條微操作，解碼器本身就是一個小 CPU。這就是 RISC 哲學的完整體現：讓硬體簡單，把複雜留給編譯器。

A　附錄：速查表

A.1　控制訊號總表

訊號	布林式
Mx_A.sel	i_{15}（0 = 指令、1 = ALUout）
A.load	$\neg i_{15} \vee (i_{15} \wedge i_5)$
D.load	$i_{15} \wedge i_4$
RAM.str	$i_{15} \wedge i_3$
RAM.addr	A[14..0]
Mx_Y.sel	i_{12}（0 = A、1 = M）
ALU 控制	$(zx..no) = (i_{11}..i_6)$
jump	$(i_2 \wedge ng) \vee (i_1 \wedge zr) \vee (i_0 \wedge \neg zr \wedge \neg ng)$
PC.load	$i_{15} \wedge jump$；PC.in = A
PC.inc	恆 1（優先序 reset > load > inc）

A.2　C-指令編碼總表

格式： `111` $a\, c_1c_2c_3c_4c_5c_6\, d_1d_2d_3\, j_1j_2j_3$

a=0	a=1	c_1	c_2	c_3	c_4	c_5	c_6
`0`		1	0	1	0	1	0
`1`		1	1	1	1	1	1
`-1`		1	1	1	0	1	0
`D`		0	0	1	1	0	0
`A`	`M`	1	1	0	0	0	0
`!D`		0	0	1	1	0	1
`!A`	`!M`	1	1	0	0	0	1
`-D`		0	0	1	1	1	1
`-A`	`-M`	1	1	0	0	1	1
`D+1`		0	1	1	1	1	1
`A+1`	`M+1`	1	1	0	1	1	1
`D-1`		0	0	1	1	1	0
`A-1`	`M-1`	1	1	0	0	1	0
`D+A`	`D+M`	0	0	0	0	1	0
`D-A`	`D-M`	0	1	0	0	1	1
`A-D`	`M-D`	0	0	0	1	1	1
`D&A`	`D&M`	0	0	0	0	0	0
`D|A`	`D|M`	0	1	0	1	0	1

dest	d_1	d_2	d_3
—	0	0	0
`M=`	0	0	1
`D=`	0	1	0
`DM=`	0	1	1
`A=`	1	0	0
`AM=`	1	0	1
`AD=`	1	1	0
`ADM=`	1	1	1

jump	j_1	j_2	j_3
—	0	0	0
`JGT`	0	0	1
`JEQ`	0	1	0
`JGE`	0	1	1
`JLT`	1	0	0
`JNE`	1	0	1
`JLE`	1	1	0
`JMP`	1	1	1

A.3　煙霧測試指令的手工組譯

以 `D=D+M` 為例示範一次完整推導：C-指令 → `111`；用到 M → $a = 1$；comp `D+M` → $c = 000010$；dest `D` → $d = 010$；無跳躍 → $j = 000$。串起來：`1111 0000 1001 0000` = `0xF090`。

指令	二進位	hex
@21	0000000000010101	0015
D=A	1110110000010000	EC10
M=D	1110001100001000	E308
D=D+M	1111000010010000	F090
M=M+1	1111110111001000	FDC8
D;JGT	1110001100000001	E301
0;JMP	1110101010000111	EA87

A.4 記憶體配置（本作業的 64KB 版本）

位址	用途
0x0000–0x3FFF	一般 RAM （真 Hack 的全部 RAM）
0x4000–0x5FFF	螢幕映射區（本作業：普通 RAM，可觀察位元圖案）
0x6000	鍵盤映射（手動編輯模擬按鍵）
0x6001–0x7FFF	真 Hack 不存在；行為未定義，隨便它

A.5 參考資料

- COMSM1302 第七週講義與影片（Hack ISA、機器碼編碼、CPU 架構輪廓、管線化）, University of Bristol.
- Nisan & Schocken, *The Elements of Computing Systems*（nand2tetris）, Ch. 5（Computer Architecture）——CPU ／ Memory ／ Computer 晶片的官方規格.
- Logisim Library Reference——RAM 的 Data Interface 屬性（“Separate load and store ports”）.
- 第二週作業（ALU）、第四週作業（PC）、第五週作業（Collatz 程式）——本作業的三塊積木.